CIRUS: an elastic cloud-based framework for Ubilytics

Linh Manh Pham, Ahmed El Rheddane, Didier Donsez, Noel de Palma

▶ **To cite this version:**

Linh Manh Pham, Ahmed El Rheddane, Didier Donsez, Noel de Palma. CIRUS: an elastic cloud-based framework for Ubilytics. Annals of Telecommunications - annales des télécommunications, 2016, 71, pp.133-140. 10.1007/s12243-015-0489-0 . hal-01245930

HAL Id: hal-01245930
https://hal.science/hal-01245930

Submitted on 17 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

CIRUS: An elastic Cloud-based framework for Ubilytics

Linh Manh Pham · Ahmed El-Rheddane · Didier Donsez · Noel de Palma

Received: date / Accepted: date

Abstract The Internet of Things (IoT) has become a reality with the availability of chatty embedded devices. The huge amount of data generated by things must be analysed with models and technologies of the "Big Data Analytics", deployed on Cloud platforms. The CIRUS project aims to deliver a generic and elastic cloud-based framework for Ubilytics (ubiquitous big data analytics). The CIRUS framework collects and analyses IoT data for Machine to Machine services using Component-off-the-Shelves (COTS) such as IoT gateways, Message brokers or Message-as-a-Service providers and Big Data analytics platforms deployed and reconfigured dynamically with Roboconf. In this paper, we demonstrate and evaluate the genericity and elasticity of CIRUS with the deployment of an Ubilytics use case using a real dataset based on records originating from a practical source.

Keywords Big Data analytics · Cloud computing · Elasticity · Internet of Things · Middleware · Ubilytics

1 Introduction

The Internet of Things (IoT) [1] has become a reality with the availability of chatty embedded devices such as RFID, wireless sensors, mobile sensors, personal smartphones, etc. According to IDC, the installed base of the Internet of Things will be approximately 212 billion "things" globally by the end of 2020 [2]. IoT provides companies with new opportunities of economic models (e.g. pay as you go), improves the quality of service delivered to their customers both individuals and companies and helps them to satisfy their legal and contractual duties. The IoT is now an established efficiency and productivity tool for agile enterprises and organizations. However, associated services called

Linh Manh Pham · Ahmed El-Rheddane · Didier Donsez · Noel de Palma
University of Grenoble Alpes, France
E-mail: first.last@imag.fr

Machine to Machine (i.e. M2M) require the analysis of a huge amount of data produced by swarms of sensors and collected by dedicated gateways. This huge amount of IoT data must be analysed with models (e.g. Map-Reduce, ESP, CEP [4]) and technologies (e.g. Hadoop, Storm [7]) of the "Big Data Analytics" (BDA for short), deployed on Cloud computing platforms [5]. This new field is named Ubilytics (ubiquitous big data analytics) by [3]. Currently, deploying Ubilytics infrastructures/projects often requires engineers with skills in various technologies of Cloud computing, Big Data, IoT as well as diverse business domains such as smart grid, supply chain, healthcare, etc. Gartner forecasts that the business behind Big Data will globally create 4.4 million IT jobs by 2015 with the mobile and social networks [23]. Ubilytics will create a new kind of IT job, the Ubilylist. Ubilylists are business and domain experts who are able to design and deploy complex analytic workflows and to interpret ubiquitous big data results. Whereas, deploying and configuring (i.e. D&C process) the infrastructure for Ubilytics is a complex task and requires several IT skills to deal with various technologies from embedded IoT gateways to cloud-based analytic platforms.

The CIRUS framework is a generic and elastic cloud-based Platform as a service (PaaS) for real-time Ubilytics. The CIRUS project targets the ubilylists who are not Cloud D&C expert. It allows them to design complex workflows of standard analytics, sensor data sources and data viz interfaces as well as to deploy these components both on the IoT gateways and on virtual servers hosted by one or several Cloud platforms. Moreover, the CIRUS PaaS aims to design and deploy rapidly Software as a service (SaaS) for Ubilytics. In summary, we make the following contributions in this paper: (1) We propose the novel architecture of CIRUS, a generic and elastic cloud-based PaaS platform for Ubilytics using Component-off-the-Shelves (COTS); (2) We perform an experiment which validates the genericity and elasticity of CIRUS using a practical Ubilytics use case. This experiment is deployed on a hybrid Cloud environment consisting of a private VMware vSphere [28] hosting center, the public Amazon EC2 [13] and Microsoft Azure [14] Clouds.

The paper is organized as follows. Section 2 presents the related work and motivation. Section 3 describes the overall architecture and components of CIRUS. The proof-of-concept implementation of CIRUS using a dataset from practical sources is demonstrated in an use case of Section 4. Section 5 uses this use case in an experiment to validate the proposed framework. Conclusions are stated in Section 6.

2 Related work and motivation

Since CIRUS framework is a flexible cloud-based PaaS platform for Ubilytics, we compare it with cloud-based BDA platforms for IoT.

On this aspect, several research works target special IoT application domains. Ma et al. [18] integrate the on-demand ability of the Cloud to their Big Data framework for IoT. In order to deal with the large volume of IoT data,

they propose an update and query efficient index framework (UQE-Index) based on key-value store that can support high insert throughput and provide efficient multi-dimensional query simultaneously. In [16], Talia introduces Data Mining Cloud Framework as a Data Analytic (DAPaaS), a programming environment for development of BDA applications, including supports for single-task and parameter-sweeping data mining programs. A cloud-based platform for BDA in power grid, validated in a micro-grid at USC campus, has been proposed by Simmhan, et al. [17]. This platform provides a pipeline mechanism to integrate information from various sources, digest and store them in static repositories for sharing knowledge. These approaches do not mention the D&C process of such complex Big Data frameworks. It is an error-prone and complicated work for the ubilylists, which needs to be automated by intermediate systems or middlewares like CIRUS does.

Only a few projects such as Globus Provision [19] and CloudMan [20] take the issue of automating deployment of BDA applications on Cloud infrastructure into consideration. Globus Provision is a cloud resource management tool for deploying biologic computational Grids on EC2. It offers tools such as CRData and Globus Transfer to deal with massive dataset on the Cloud. CloudMan is also a Cloud resource management tool for automation of the deployment of a Galaxy instance on EC2. However, these two frameworks are dedicated to one specific Cloud and used for only one type of BDA applications. In contrast to CloudMan and Globus Provision, Jin, et al. [21] propose an automatic deployment framework which supports heterogeneous environments for deployment of BDA applications. However, it only supports static deployment, the elasticity is not considered at all. Elasticity is one of the major benefits of Cloud computing to add quality of service (QoS) into cloud services [15]. To the best of our knowledge, CIRUS is the first preliminary framework providing automated D&C and elasticity for cloud-based BDA applications.

Along with academic efforts, commercial Big Data as-a-Service (BDaaS) solutions for IoT are currently available and built on open-source platforms. Splunk Cloud [25] is a proprietary cloud-based service that consumes, searches and visualizes different sources of machine data. This multi-tenant DA SaaS model provides analysing and monitoring suites for developers, application support staffs, system administrators, and security analysts who have to deal with a bulk of data every single day. Amazon Kinesis offers real-time processing, scalable data throughput and volume which accommodate up to "hundreds of terabytes of data per hour from hundreds of thousands of sources" [24]. However, those solutions are not portable from one provider to another. Interoperability, portability and hybrid cloud configuration are seriously compromised. Moreover, although these Big Data IoT systems are implemented and orchestrated automatically on the Cloud, they do not take into account the dynamic reaction to changes from surrounding environment.

In comparison with the state-of-the-art, the CIRUS project aims to provide a generic and elastic PaaS Cloud infrastructure for BDA, which digests IoT

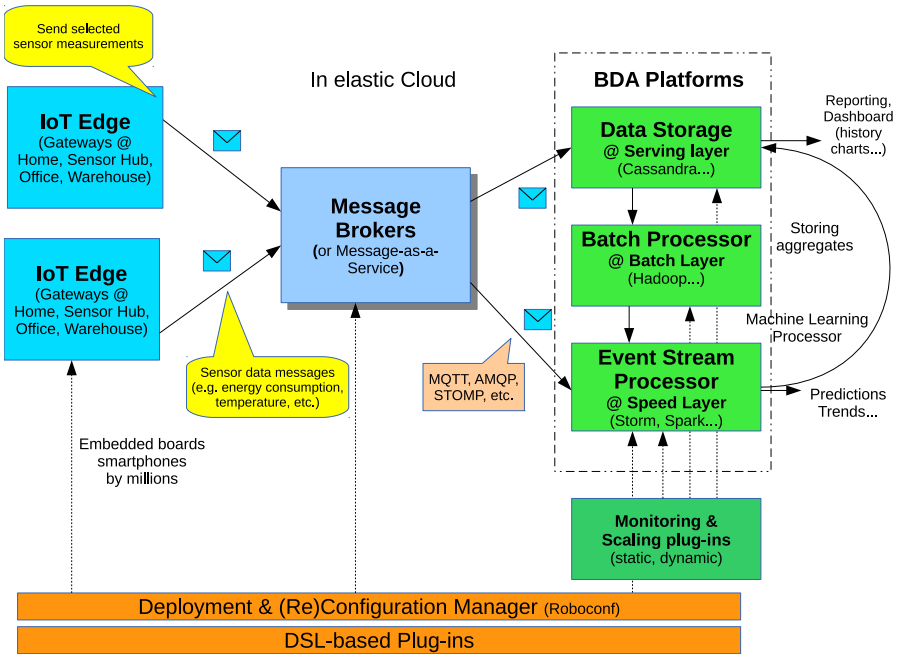

Fig. 1: The CIRUS novel framework implementing lambda architecture

data and covers the automated D&C process of software components on both IoT gateways and Cloud machines.

3 The CIRUS framework

The CIRUS framework is designed as a flexible architecture described as a set of abstract components. Each of these abstract components will be specialized into concrete components when a CIRUS application is deployed on the Cloud. Therefore the platform assembly can be replaced easily to use new IoT gateway protocols, message brokers and BDA platforms. The overall novel architecture of the CIRUS framework is depicted in Fig. 1. As shown in this figure, CIRUS is a combination of 3 application tiers which are IoT edges, Message brokers and BDA platforms. The BDA platform tier, in turn, is divided into 3 main layers (serving, batch and speed) according to Lambda architecture [26] which is a data-processing architecture designed to deal with massive data from multiple sources. A lambda system contains typically three layers: batch processing aiming to perfect accuracy, speed (or real-time stream processing) for minimizing latency, and a serving layer for responding to queries. Following is details about main components of CIRUS.

IoT Edge tier is deployed on the IoT gateways. It collects data from various sensor networks (e.g. enOcean, Zigbee, etc.) and publish them to the

brokers. They store temporally the data when the network such as Wifi public hotspots or ADSL is not available. Gateways can be either mobile (from people, cars to sounding balloons) or static (vending machines, interactive digital signages). **Message Brokers** tier implements the distributed publish-subscribe patterns with various QoS properties. These include fault-tolerance, high-availability, elasticity, causality, deterministic delivery time, etc. **BDA Platforms** tier implements various BDA solutions, which comprises of following three layers:

- **Data Storage** operates at Serving layer to store temporal data of sensors as well as calculated aggregation data and prediction models.
- **Batch processor** operating at Batch layer is to retrieve offline data from the storages and perform batch processing.
- **Event stream processor** operating at Speed layer aims to analyse in real-time the stream of sensor data incoming from the IoT edges thanks to the brokers to classify the data. **Machine Learning processor** situated between batch and speed layers is to calculate unsupervised models of offline big data and parameterize the event stream processing (i.e ESP).

DSL-based plug-ins allows to describe Big Data applications in terms of abstract components that vary from the sensors to use, the clouds to implement to the topologies to reuse, etc. **D&C manager** parses the application descriptions to concrete instances which can be deployed, configured, adapted automatically in a hybrid cloud infrastructure and on physical machines (gateways) at runtime. The D&C manager allows third-party **monitoring and scaling plug-ins** to be integrated in its core. This component monitors the three lambda layers and makes scaling decisions based on variations of environment that enables the elasticity for the framework.

Based on the flexible architecture which is a composition of many COTS components, CIRUS can implement and (self-)manage any kind of BDA applications. As a first step towards validating the genericity and elasticity of CIRUS, we describe in details its selected components and data-flow processing in a smart-grid use case. CIRUS allows to use OTSO (Off-the-shelf Option) [30] and CAP (COTS Acquisition Process) [31] as COTS selection methods. With the smart-grid domain, we use CAP, which considers domain/domain compatibility criterion, to select appropriate components among many candidates for each tier as well as layer of CIRUS.

4 The smart-grid use case

For validating our Ubilytics PaaS, we have implemented a prototype of CIRUS for simple smart-grid application in which the electricity supplier can forecast the electric load in the next minutes by analysing in real time electric consumptions collected in houses. In the short term, the infrastructure will be adjusted dynamically according to the load of sensor messages' throughput by an autonomous manager such as Roboconf [12]. For instance, additional cloud

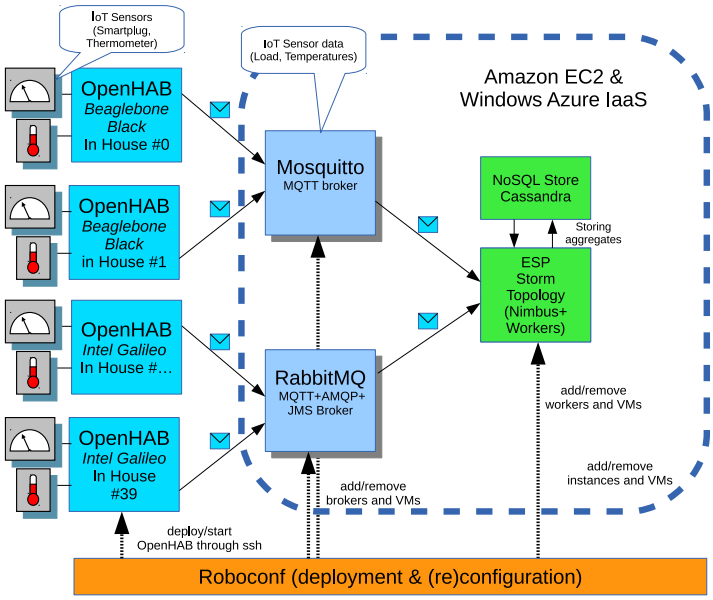

Fig. 2: The CIRUS's components for the smart-grid use case

instances can be added to the speed layer of the architecture when workload goes over a threshold. The detailed architecture of the CIRUS prototype for the use case is shown in Fig. 2. As the batch processing layer is not necessary in this particular use case, without loss of generality, we have not implemented it in the prototype. The components of the smart-grid use case are described as follows.

4.1 IoT gateways

For the sensor data collection, we have chosen the OpenHAB framework [1] which provides an integration platform for sensors and actuators of the home automation. OpenHAB allows users to specify DSL-based rules which will be parsed by its rule engine to update the actuator's commands upon sensor's state changes. The Event-Condition-Action (ECA) paradigm is used by OpenHAB for executing the home automation actions. ECA rules are triggered on sensor value changes, command emission and timer expiration. Events (e.g. state changes and commands) can be "imported" or "exported" using bindings for MQTT, XMPP, Twitter, etc. OpenHAB can be installed and run on

[1] Now is ESH [8]

embedded boards, some of which are Raspberry Pi, Beaglebone Black and Intel Galileo. For the smart-grid use case, we have developed a new OpenHAB plug-in (called binding) in order to replay the sensor log files containing the smart-plug measurements (e.g. timestamped load and work) of each house. It is deployed on both embedded boards and virtual machines (VMs) on the Cloud with one instance per house. Data from OpenHAB is sent to the MQTT brokers.

4.2 MQTT brokers

Message Queue Telemetry Transport (MQTT) [9] is a transport data protocol for M2M networks. It is devised for supporting low-bandwidth and unreliable networks, as illustrated by satellite links or sensor networks. MQTT follows the publish-subscribe pattern between sensors and one or more sinks like M2M gateways, etc. Mosquitto [10] and RabbitMQ [11] are the main robust and open-source implementations of MQTT brokers that we have selected for this use case. The data from sensors are classified in the brokers according to specific topics (e.g. houses or plugs) and then delivered to the Speed layer.

4.3 Speed layer for real-time analytics

For the speed layer of the lambda architecture, we have chosen the Apache Storm framework. Storm is a real-time ESP system. It is designed to deploy a processing chain in a distributed infrastructure such as a Cloud platform (IaaS). For the Ubilytics platform, we have developed a new Storm input components (called spout) in order to generate sensor tuples from the MQTT brokers by subscribing on the MQTT topics with one spout per house.

4.4 Historical data storage

In the speed layer, the Storm topology needs to maintain some execution ongoing state. This is the case for the sliding window average of sensor values. To do this we use Storm with Cassandra [22] for our real-time power consumption prediction. Cassandra is an open source distributed database management system (NoSQL solution). It is created to handle large amounts of data spread out across many nodes, while providing a highly available service with no single point of failure. Cassandra's data model allows incremental modifications of rows.

4.5 D&C manager

As mentioned, Roboconf is our choice for the elastic D&C manager in multi-Clouds. It is a middleware for autonomously configuring, installing, and managing complex legacy application stacks deployed on the cloud VMs and on

```
# An Azure VM
VM_AZURE {
  alias: VM Azure;
  installer: iaas;
  children: Storm_Cluster,
            Cassandra;
}

# A BeagleBone Black
BOARD_BEAGLEBONE {
  alias: BeagleBone Black;
  installer: embedded;
  children: OpenHAB;
}
```

```
# Storm Cluster for ESP
Storm_Cluster {
  alias: Storm Cluster;
  installer: bash;
  imports: Nimbus.port, Nimbus.ip;
  children: Nimbus, Storm_Supervisor;
}

# OpenHAB: A Home Automation Bus
OpenHAB {
  alias: OpenHAB;
  installer: puppet;
  exports: ip,
       brokerChoice = Mosquitto;
  imports: Mosquitto.ip,
       Mosquitto.port;
}
...
```

Fig. 3: Components of the smart-grid use case under Roboconf's DSL

physical machines which dynamically evolve over time. Following the Service-Oriented Architecture (SOA) principles, Roboconf consists of a simple DSL for describing configuration properties and inter-dependencies of service components; a distributed configuration protocol ensuring the dynamic resolution of these inter-dependencies; a runtime system that guarantees the correct (re)deployment and management of distributed application across multiple Clouds, physical devices or virtual machines. Roboconf supports popular IaaS Cloud platforms such as EC2, Azure and vSphere. The configuration of components in smart-grid use case under Roboconf's DSL are excerpted and shown in Fig. 3.

This section describes the prototype for the smart-grid use case which serves as a proof-of-concept for the genericity of CIRUS. However, to validate the elasticity, an experiment in an autonomous environment needs to be conducted.

5 Validating experiments

In order to evaluate the elasticity of the proposed framework, we use the smart-grid prototype in implementing an experiment which worker instances of "Event stream processor" component (the Storm cluster in this situation) will be scaled out/in depending on fluctuation of workload. The experiment is deployed on a hybrid Cloud environment. Each house is represented by an OpenHAB process which publishes a given dataset related to the house (approximately 2.5 GB per house). The dataset files, which are derived from DEBS 2014 Grand Challenge [6], are preloaded on microSD cards. The dataset is based on practical records collecting from smart plugs, which are deployed in private households of 40 houses located in Germany. Those data are collected

```
# A VM Azure with Nimbus
instanceof VM_AZURE {
  name: vm-azure-nimbus-1;

  instanceof Nimbus {
    name: nimbus-storm-1;
    port: 6627;
  }
}

# A BeagleBone Board for OpenHAB
instanceof BOARD_BEAGLEBONE {
  name: board-bb-openhab-1;

  instanceof OpenHAB {
    name: openhab-1;
  }
}
```

```
# A VM EC2 for Message Broker
instanceof VM_EC2 {
  name: vm-ec2-mosquitto-1;

  instanceof Mosquitto {
    name: mosquitto-1;
    port: 1883;
  }
}

# A VM vSphere with Cassandra
instanceof VM_VSPHERE {
  name: vm-vsphere-cassandra-1;

  instanceof Cassandra {
    name: cassandra-1;
  }
}
...
```

Fig. 4: Instances of components of the smart-grid use case under Roboconf's DSL

roughly every second for each sensor in each smart plug. It is worth noting that the data set is gathered in an real-world, uncontrolled environment, which implies the possibility of producing imperfect data and measurements.

The OpenHAB gateways are deployed on Beaglebone Black [27] embedded boards. The aggregated data and results are finally stored in an Apache Cassandra database which is in our vSphere private Cloud for safety. The cluster of MQTT brokers, containing the topics, are hosted on EC2 public Cloud VMs. The Storm cluster contains worker nodes, each corresponding to a VM instance on Azure Cloud for taking advantage of the computing strength of our granted Microsoft infrastructure. The dynamic D&C process is performed automatically by Roboconf. Fig. 4 shows a short extract about instances of the experiment under Roboconf's DSL.

5.1 Storm topology for real-time load forecasting

To handle the consumption prediction query in a continuous and scalable manner we use a Storm-based solution. The elaborations consist of queries that are continuously evaluated on the events supplied as input. Storm cluster is composed of two kinds of nodes: a master node which runs a daemon called Nimbus and worker ones which run a daemon called Supervisor. The Nimbus is responsible for distributing code around the cluster, assigning tasks to machines and monitoring for failures. Each supervisor listens for works assigned to its node, it starts and stops worker processes when necessary based on what Nimbus has assigned to it. Each worker process executes a subset of a topology; a running topology may consist of many worker processes spread across many nodes.

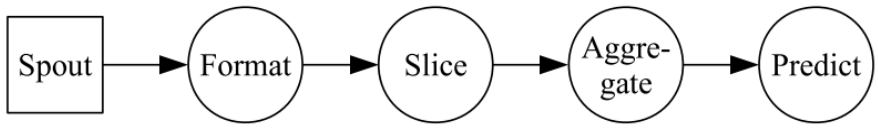

Fig. 5: The Storm topology used in the smart-grid use case

A topology is a graph of computation. Each node in a topology encompasses a processing logic, and links between nodes indicate data flows between these ones. The unit of information that is exchanged among components is referred to as a tuple, which is a named list of values. There are two types of components which are spout and bolt. A spout component encapsulates the production of tuples. A bolt encapsulates a processing logic manipulating the tuples such as filtering. The communication patterns among components are represented by streams, unbounded sequences of tuples that are emitted by spouts or bolts and consumed by bolts. Each bolt can subscribe to many distinct streams in order to receive and consume their tuples. Both spouts and bolts can emit tuples on different streams as needed. The diagram in Fig. 5 presents our Storm topology for electric load prediction, which is submitted to Storm clusters for execution. The spout retrieves the data from an MQTT topic where it is published line by line, these lines are then formatted into a list of values by the Format bolt. Then, only the load-related tuples are filtered and enriched with a time slice based on their timestamps and aggregated per house and time slice. Prevision is then made given the recent time slice history and previous time slices which are stored in a Cassandra database.

5.2 Monitoring and scaling decision

We have developed a Ganglia [29] plug-in for Roboconf to monitor the topology and fetch the metrics that we consider relevant to characterize the load of a given component. Moreover, Storm also exposes application-metrics, such as the number of emitted or executed tuples. We consider that these Storm metrics are the most fit to reflect the load of a Storm component. Basically, the state of a component is described by the difference between the tuples it receives, i.e., the tuples that are emitted by all its input components, and the tuples it is able to process within a given period of time. While this is straight forward to compute, it takes into account virtually any type of bottlenecks a component might encounter: should the processing of a tuple be CPU-intensive or should it rely on an external web service or database connection, it would invariably tell if a given component is able to keep up with the pace of its incoming stream of data.

Once we have the Storm metrics of the components and provided that load balancing among different instances of the same component is guaranteed, through a random policy for instance, the scaling decision goes as follows:

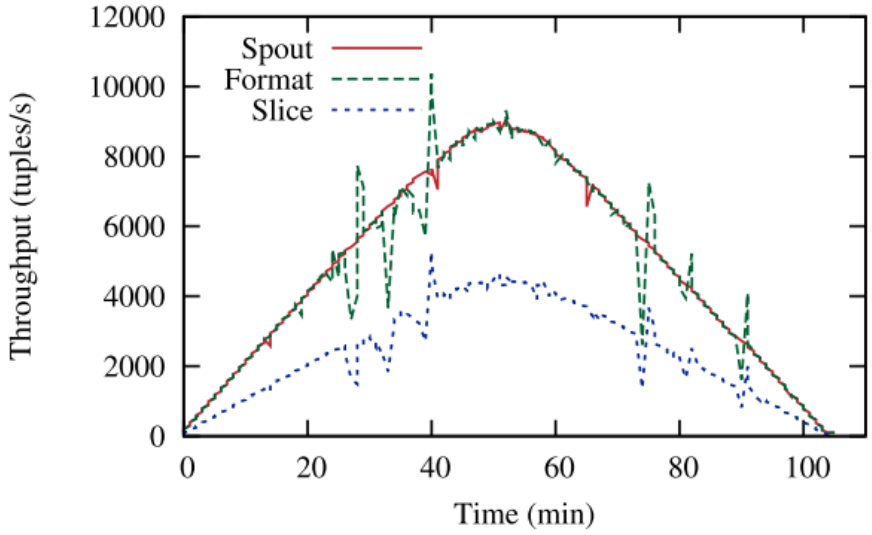

Fig. 6: Throughput per component

– Scaling out: which is the addition of an extra component instance, should occur when a given component receives more messages than it is able to process, i.e., if formula (1) is verified, n being the current number of the component's instances.

$$\sum_{inputs} emitted > \sum_{i<n} executed_i \tag{1}$$

Before scaling out, we store the last witnessed capacity of the component's instances as described by equation (2). This will be later used to initiate scaling in.

$$capacity = \frac{\sum_{i<n} executed_i}{n} \tag{2}$$

– Scaling in: which is the removal of an unnecessary component instance, should take place if we can guarantee that, with one instance less, the component's instances would still not be overloaded. Using the previously stored capacity, that is updated upon each scaling out, scaling in is instigated upon formula (3) verification.

$$\sum_{inputs} emitted > (n-1)capacity \tag{3}$$

Obviously, changing the number of component instances would not have the desired effect on performance unless it is coupled with a correspondent change in the amount of provisioned resources.

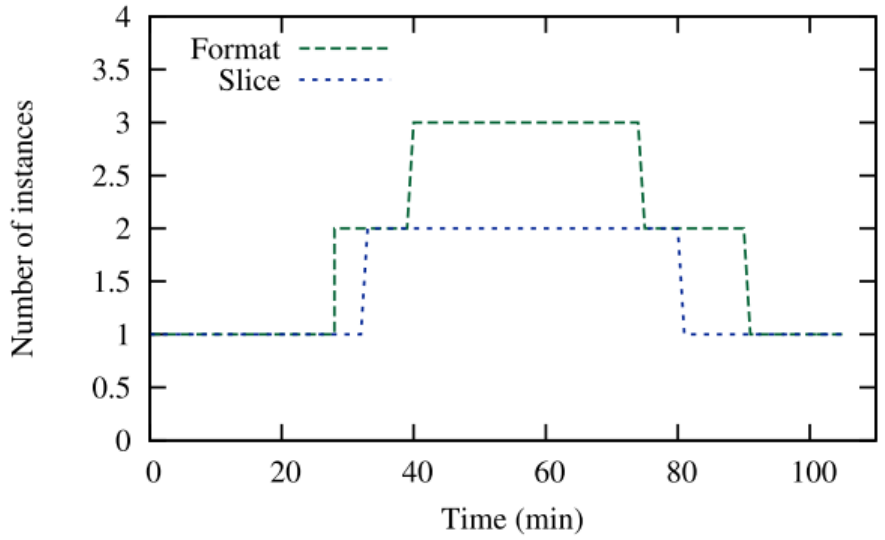

Fig. 7: Parallelism per component

5.3 Result discussion

This experiment shows how our framework provides elasticity for BDA applications in multi-Clouds using the prototype for smart-grid use case. The initial cluster is set so as to have one worker per component of our topology. The incoming throughput is then varied gradually in order to see its impact on the behaviour of our topology.

Figures 6 and 7 show the results of this experiment, components that have not needed scaling have been omitted. We can see that at times, there is a decrease in the number of emitted bolts (Fig. 6). This is detected by our elasticity controller which instigate the corresponding scaling operation (Fig. 7). The pikes following each scaling are due to accumulated non executed tuples when scaling is carried out. Likewise, when the spout's throughput is low enough our controller removes unnecessary instances.

Note that different components are scaled separately. Format has been the first to be scaled, and as it became able to process all the tuples its received, it emitted more tuples which resulted in Slice being overloaded later on. Slice's throughput is roughly half the throughput of the previous components as it filters only load related measurements which accounts for half the total reported measurements. Thanks to the elasticity, our framework has been able to efficiently support the variations of dataload.

6 Conclusion

This paper demonstrates the CIRUS project in the aspects of an Ubilytics novel generic architecture and its elastic implementation for a smart-grid use case. The CIRUS project aims to deliver a self-adaptive cloud-based framework for Ubilytics applications. The CIRUS framework collects and analyses IoT data for M2M services using COTS such as IoT gateways, Message brokers or Message-as-a-Service providers and BDA platforms. For the smart-grid use case, CIRUS deploys OpenHAB processes for house gateways, several MQTT brokers and a Storm topology which analyses the sensor data stream of the smart-grid application for power prediction. The experiment validated the genericity and elasticity of CIRUS's architecture.

References

1. Harald Sundmaeker, Patrick Guillemin, Peter Friess, Vision and Challenges for Realising the Internet of Things, Publications Office of the European Union, Luxembourg, 2010.
2. IDC, Worldwide and Regional Internet of Things (IoT) 2014–2020 Forecast: A Virtuous Circle of Proven Value and Demand, May 2014, http://www.idc.com/getdoc.jsp?containerId=248451
3. Niklas Elmqvist, Pourang Irani, "Ubiquitous Analytics: Interacting with Big Data Anywhere, Anytime", Computer, vol.46, no. 4, pp. 86-89, April 2013.
4. James Manyika et al., "Big data: The next frontier for innovation, competition, and productivity," McKinsey Global Institute, Tech. rep. May 2011.
5. Ignacio M. Llorente, "Key Challenges in Cloud Computing to Enable Future Internet of Things," The 4th EU-Japan Symposium on New Generation Networks and Future Internet, January 2012.
6. DEBS GC 2014, 'http://www.cse.iitb.ac.in/debs2014/?page_id=42," visited on July 2015.
7. Storm, 'http://storm-project.net/," visited on July 2015.
8. ESH, 'http://www.eclipse.org/smarthome/," visited on July 2015.
9. MQ Telemetry Transport, 'http://mqtt.org," visited on July 2015.
10. Mosquitto, 'http://mosquitto.org/," visited on July 2015.
11. RabbitMQ, 'http://www.rabbitmq.com/," visited on July 2015.
12. Pham, Linh Manh; Tchana, Alain; Donsez, Didier; de Palma, Noel; Zurczak, Vincent; Gibello, Pierre-Yves, "Roboconf: A Hybrid Cloud Orchestrator to Deploy Complex Applications," in Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on , vol., no., pp.365-372, June 27 2015-July 2 2015
13. Amazon EC2, 'http://aws.amazon.com/ec2/," visited on July 2015.
14. Microsoft Azure, 'http://www.windowsazure.com," visited on July 2015.
15. Coutinho E.F., de Carvalho Sousa F.R., Rego P.A.L., Gomes D.G., Souza J.N., Elasticity in cloud computing: a survey, pp. 1-21, Ann. Telecommun, 2014.
16. Talia, D., "Clouds for Scalable Big Data Analytics," Computer , vol.46, no.5, pp.98,101, May 2013.
17. Simmhan, Y.; Aman, S.; Kumbhare, A.; Rongyang Liu; Stevens, S.; Qunzhi Zhou; Prasanna, V., "Cloud-Based Software Platform for Big Data Analytics in Smart Grids," in Computing in Science & Engineering , vol.15, no.4, pp.38-47, July-Aug. 2013.
18. Youzhong Ma, Jia Rao, Weisong Hu, Xiaofeng Meng, Xu Han, Yu Zhang, Yunpeng Chai, and Chunqiu Liu. 2012. An efficient index for massive IOT data in cloud environment. In Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM '12). ACM, New York, NY, USA, 2129-2133.
19. Bo Liu; Sotomayor, B.; Madduri, R.; Chard, K.; Foster, I., "Deploying Bioinformatics Workflows on Clouds with Galaxy and Globus Provision," in High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion: , vol., no., pp.1087-1095, 10-16 Nov. 2012

20. Afgan E, Baker D, Coraor N, Chapman B, Nekrutenko A, Taylor J. Galaxy CloudMan: delivering cloud compute clusters. BMC Bioinformatics. 2010; 11 Suppl 12:S4.
21. Chao Jin; Wenjun Wu; Hui Zhang, "Automating Deployment of Customized Scientific Data Analytic Environments on Clouds," in Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on , vol., no., pp.41-48, 3-5 Dec. 2014
22. Apache Cassandra, 'http://cassandra.apache.org," visited on July 2015.
23. Gartner says big data creates big jobs: 4.4 million IT jobs globally to support big data by 2015. 'http://www.gartner.com/newsroom/id/2207915," visited on July 2015.
24. Amazon Kinesis, 'https://aws.amazon.com/kinesis/," visited on July 2015.
25. SplunkCloud, 'http://www.splunk.com/en_us/products/splunkcloud.html," visited on July 2015.
26. Lambda Architecture: A state-of-the-art. 'http://www.datasalt.com/2014/01/lambda-architecture-a-state-of-the-art/," visited on July 2015.
27. Beaglebone Black, 'http://beagleboard.org/bone," visited on July 2015.
28. VMware vSphere, 'https://www.vmware.com/products/vsphere," visited on July 2015.
29. Ganglia, 'http://ganglia.sourceforge.net," visited on July 2015.
30. J. Kontio. A Case Study in Applying a Systematic Method for COTS Selection. In Proceedings of the 18th International Conference on Software Engineering, 1996.
31. M. Ochs, D. Pfahl, G. Chrobok-Diening, and B. Nothhelfer-Kolb. A Method for Efficient Measurement based COTS Assessment and Selection - Method Description and Evaluation Results. In Proceedings of the Seventh International Software Metrics Symposium, pages 285–297, 2001.